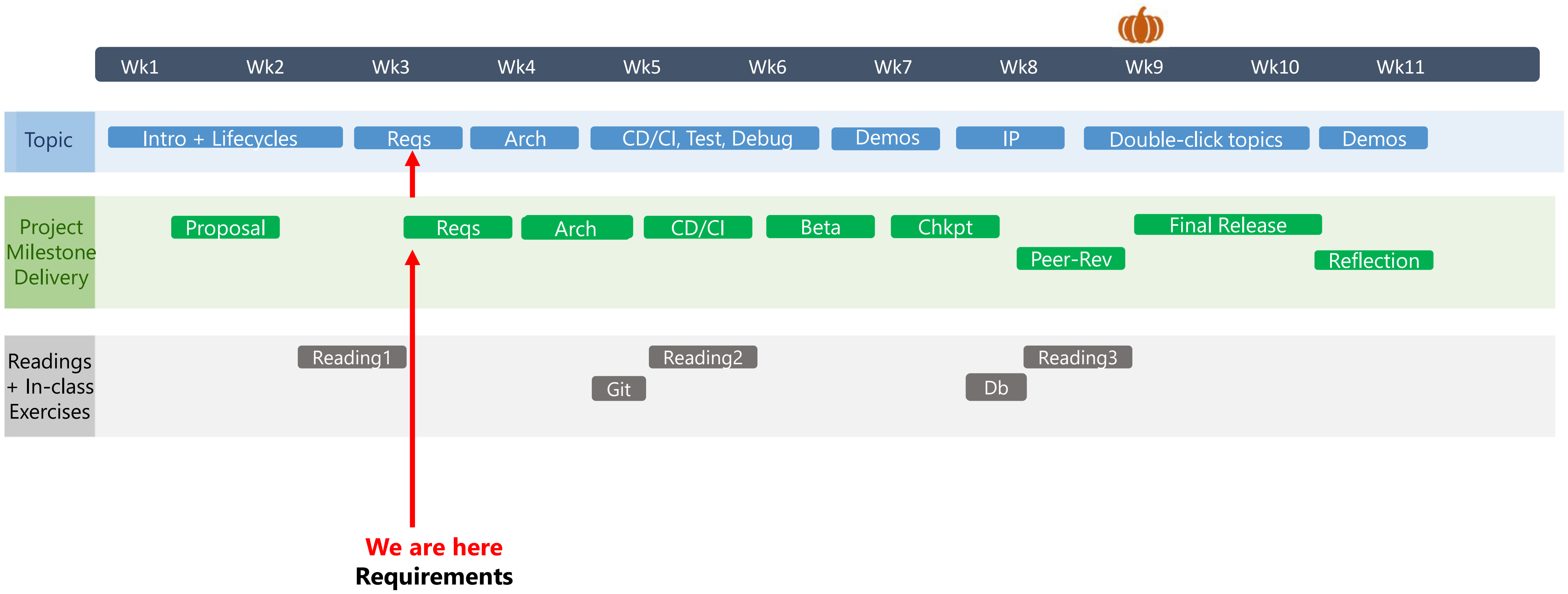
Software Requirements

CSE 403 Software Engineering

Autumn 2023

We are moving through the SDLC components



Today's Outline

1. What are requirements and what is their value?
2. How can we gather requirements?
3. What are techniques used to specify them?

Recapping where requirements fit in

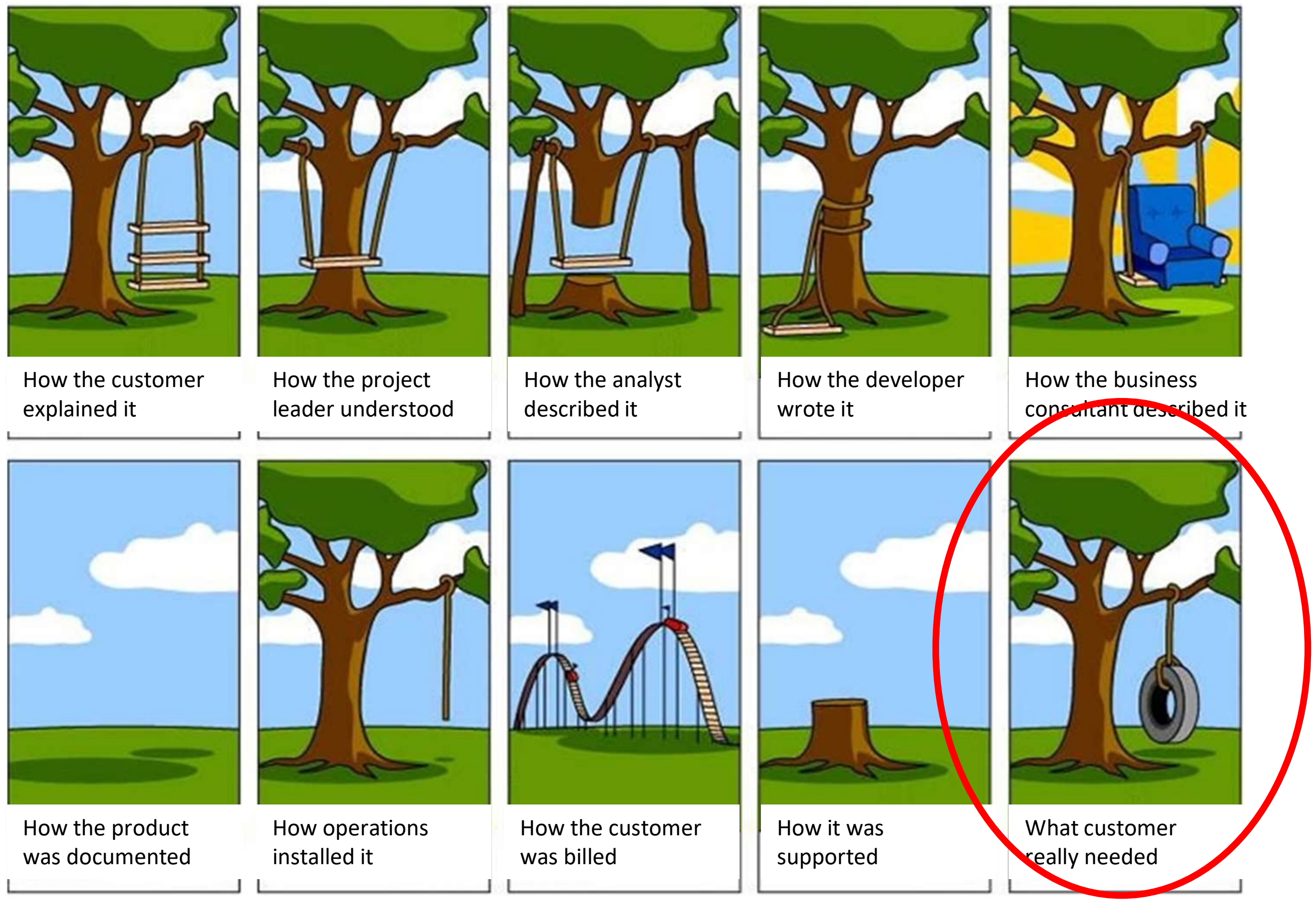
Virtually all SDLC models have the following stages →

Requirements are at the top of the list as we start the journey of product development

<u>Common stages</u>

- Requirements
- Design
- Implementation
- Testing
- Release
- Maintenance

Sharing a visual of their importance


How the customer explained it
How the project leader understood
How the analyst described it
How the developer wrote it
How the business consultant described it
How the product was documented
How operations installed it
How the customer was billed
How it was supported
What customer really needed

So, what exactly are software requirements?

Requirements specify what to build

- describe **what, not how**
- describe customer needs, not how they'll be implemented
- reflect product design, not software design

Often you need to dig deep to get a solid set of requirements

Let’s work through an example

Are these good requirements for a music player?

- Available on web and mobile
- Provide volume control
- Provide ability to flag favorites using a pulldown menu
- Enable variable playback speed
- Propose songs using ChatGPT recommendations
- Propose songs based on customer selected genres
- Written in javascript for extensibility and reliability

How about our swing example

What are good **and sufficient** requirements for the swing?

- Attaches to a single branch of a tree
- Seats one person 3-5ft tall
- Swings when pushed
- Appeals to environmental advocates
-
-

Requirements are hard but important

They help us:

- **Understand** precisely what is required of the software
- **Communicate** this understanding precisely to all involved parties
- **Monitor and control** production to ensure that system meets specification

In practice, they're used by many during SDLC

- **Customers**: what should be delivered (contractual base)
- **Project managers**: scheduling and monitoring (progress indicator)
- **Designers**: basis for a spec to design the system
- **Developers**: a range of acceptable implementations
- **QA / Testers (DevTest)**: a basis for testing, verification, and validation

Today’s Outline

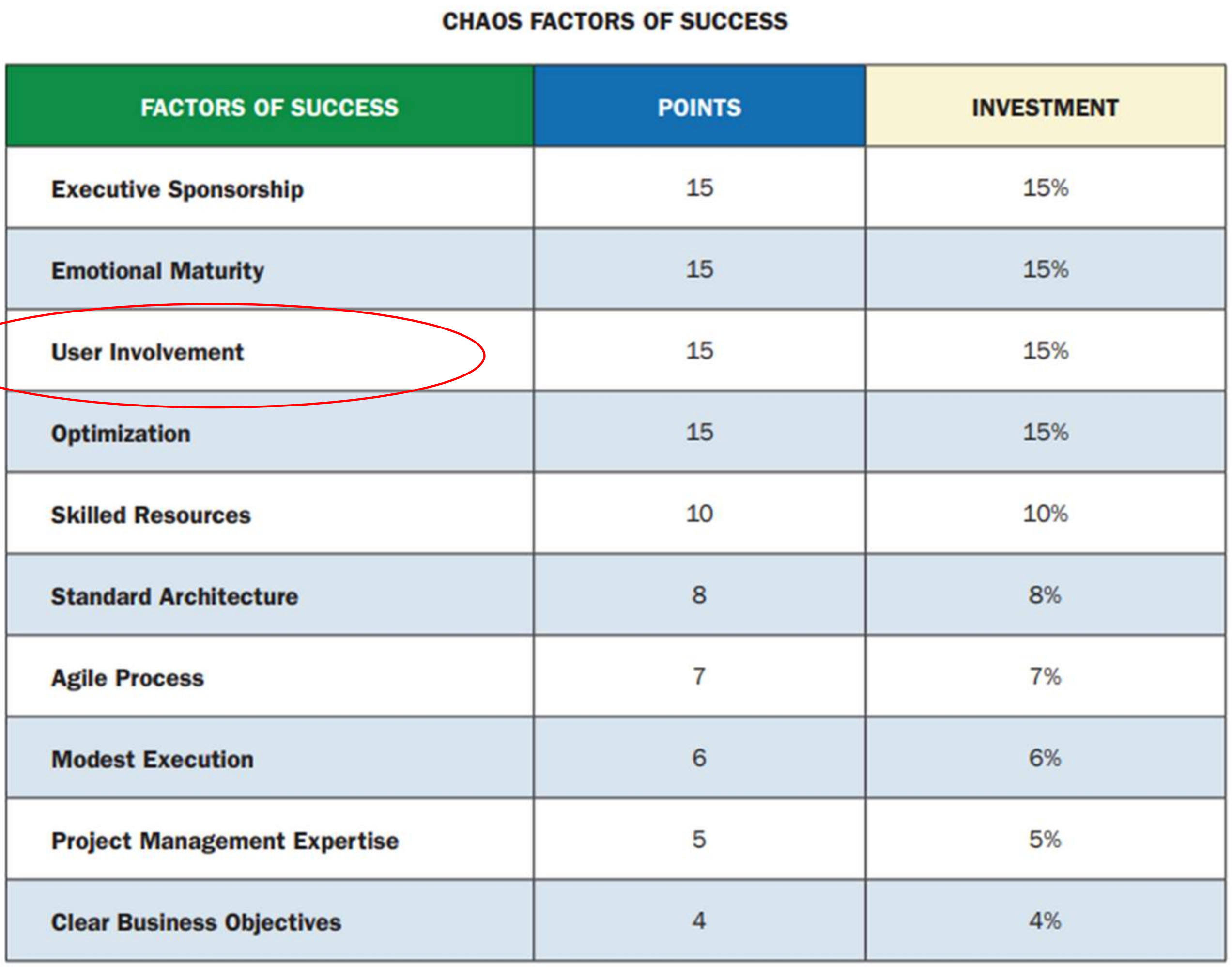
1. What are requirements and what is their value?
2. **How can we gather (elicit) requirements?** ⟵
3. What are techniques used to specify them?

Let’s start with some data

From the Standish report on software project success (2015)

Customer involvement is 3rd highest factor of project success!

https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf



CHAOS FACTORS OF SUCCESS

FACTORS OF SUCCESS	POINTS	INVESTMENT
Executive Sponsorship	15	15%
Emotional Maturity	15	15%
User Involvement	15	15%
Optimization	15	15%
Skilled Resources	10	10%
Standard Architecture	8	8%
Agile Process	7	7%
Modest Execution	6	6%
Project Management Expertise	5	5%
Clear Business Objectives	4	4%

The 2015 Factors of Success. This chart reflects our opinion of the importance of each attribute and our recommendation of the amount of effort and investment that should be considered to improve project success.

Companies recognize this – for example…

The customer is always right

(Marshall Field's department store slogan, 1852)

Customer obsession rather than competitor focus

(One of Amazon's four principles)

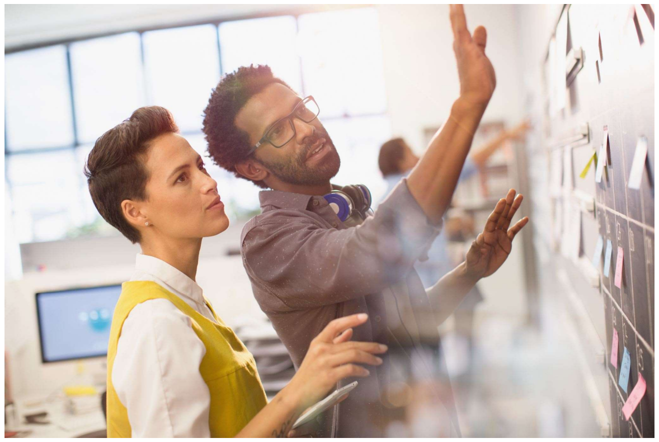
(1) Understand and serve the customer better than anyone else,
(2) forget about everything else, and
(3) make sure every little thing you do serves (1), always and everywhere

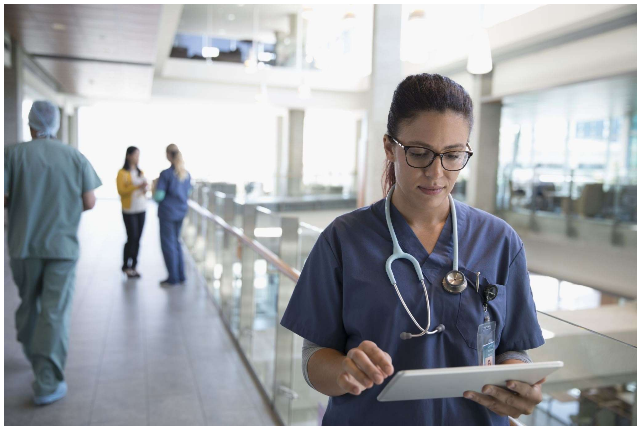
(Summary of Apple's original three principles, Steve Jobs)

Ok, so, how do we engage with customers

Ideas?

- Interviews
- Observations
- Shadowing
- Prototyping
- Mockups
- Hallway conversations
- Focus groups





Keep your customer (user) at the center of the discussion
Listen, observe and **ask clarifying questions**

Do's and don'ts in requirements gathering

Do:

- Talk to the **customers** -- to learn how they work
- Ask questions throughout the process -- "dig" for requirements
- Think about **why** users do something in your service, not just what
- Allow (and expect) requirements to change later

Do’s and don’ts in requirements gathering

Do:

- Talk to the **customers** -- to learn how they work
- Ask questions throughout the process -- "dig" for requirements
- Think about **why** users do something in your service, not just what
- Allow (and expect) requirements to change later

Don't:

- Be too specific or detailed* (caveats apply)
- Describe complex business logic or rules of the system
- Describe the exact user interface used to implement a feature
- Try to think of everything ahead of time* (caveats apply)
- Add unnecessary features not wanted by the customers

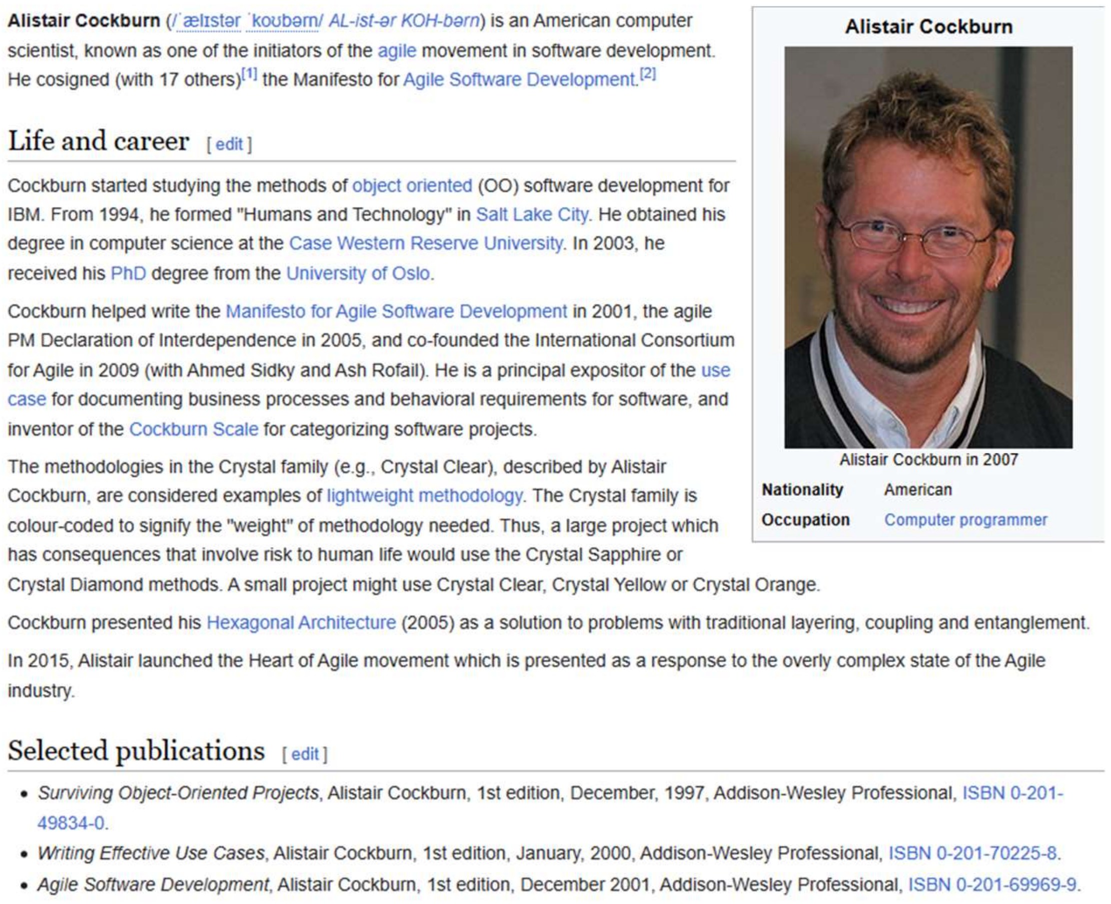
The whole process is more formally known as requirements engineering

Requirements engineering is the science of eliciting, analyzing, documenting, and maintaining requirements

As you collect your class project requirements, consider three categories:

- **Functional requirements**
  - e.g., input-output behavior
- **Non-functional requirements**
  - e.g., security, privacy, scalability
- **Additional constraints**
  - e.g., programming language, frameworks, testing infrastructure

Meet Alistair Cockburn – Requirements SME



Alistair Cockburn (/ˈælɪstər ˈkoʊbərn/ *AL-ist-ər KOH-bərn*) is an American computer scientist, known as one of the initiators of the agile movement in software development. He cosigned (with 17 others)[1] the Manifesto for Agile Software Development.[2]

Life and career [edit]

Cockburn started studying the methods of object oriented (OO) software development for IBM. From 1994, he formed "Humans and Technology" in Salt Lake City. He obtained his degree in computer science at the Case Western Reserve University. In 2003, he received his PhD degree from the University of Oslo.

Cockburn helped write the Manifesto for Agile Software Development in 2001, the agile PM Declaration of Interdependence in 2005, and co-founded the International Consortium for Agile in 2009 (with Ahmed Sidky and Ash Rofail). He is a principal expositor of the use case for documenting business processes and behavioral requirements for software, and inventor of the Cockburn Scale for categorizing software projects.

The methodologies in the Crystal family (e.g., Crystal Clear), described by Alistair Cockburn, are considered examples of lightweight methodology. The Crystal family is colour-coded to signify the "weight" of methodology needed. Thus, a large project which has consequences that involve risk to human life would use the Crystal Sapphire or Crystal Diamond methods. A small project might use Crystal Clear, Crystal Yellow or Crystal Orange.

Cockburn presented his Hexagonal Architecture (2005) as a solution to problems with traditional layering, coupling and entanglement.

In 2015, Alistair launched the Heart of Agile movement which is presented as a response to the overly complex state of the Agile industry.

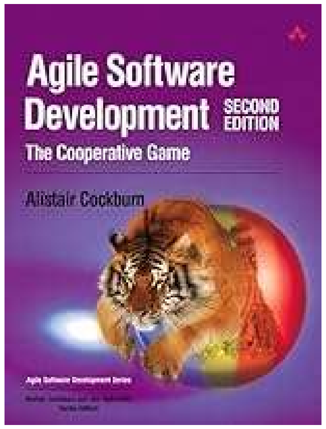
Alistair Cockburn

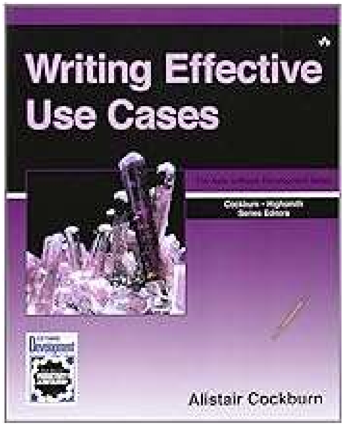
Alistair Cockburn in 2007

Nationality	American
Occupation	Computer programmer

Selected publications [edit]

- *Surviving Object-Oriented Projects*, Alistair Cockburn, 1st edition, December, 1997, Addison-Wesley Professional, ISBN 0-201-49834-0.
- *Writing Effective Use Cases*, Alistair Cockburn, 1st edition, January, 2000, Addison-Wesley Professional, ISBN 0-201-70225-8.
- *Agile Software Development*, Alistair Cockburn, 1st edition, December 2001, Addison-Wesley Professional, ISBN 0-201-69969-9.





https://en.wikipedia.org/wiki/Alistair_Cockburn

Cockburn requirements template

1. Purpose and scope
2. Terms (glossary)
3. **Use cases (the central artifact of requirements)**
4. Technology used
5. Other
   - Development process: participants, values (fast-good-cheap), visibility, competition, dependencies
   - Business rules (constraints)
   - Performance demands
   - Security, documentation
   - Usability
   - Portability
   - Unresolved (deferred)
6. Human factors (legal, political, organizational, training)

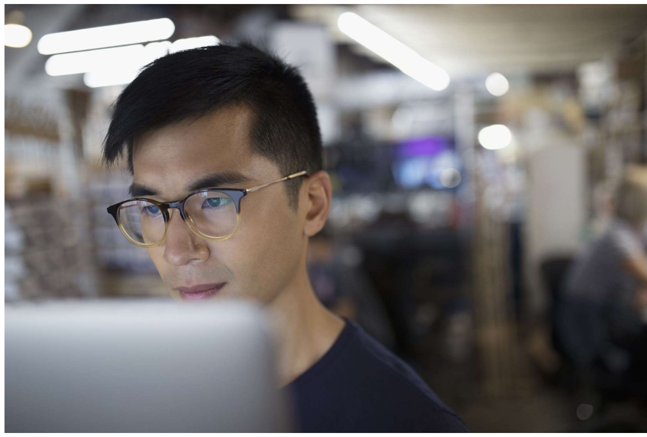
Many companies will have a template for you to use

Uniformity is good for you and the customer

See what we're asking of you in the **Requirements milestone assignment**!

More tips – watch out for these as you engineer

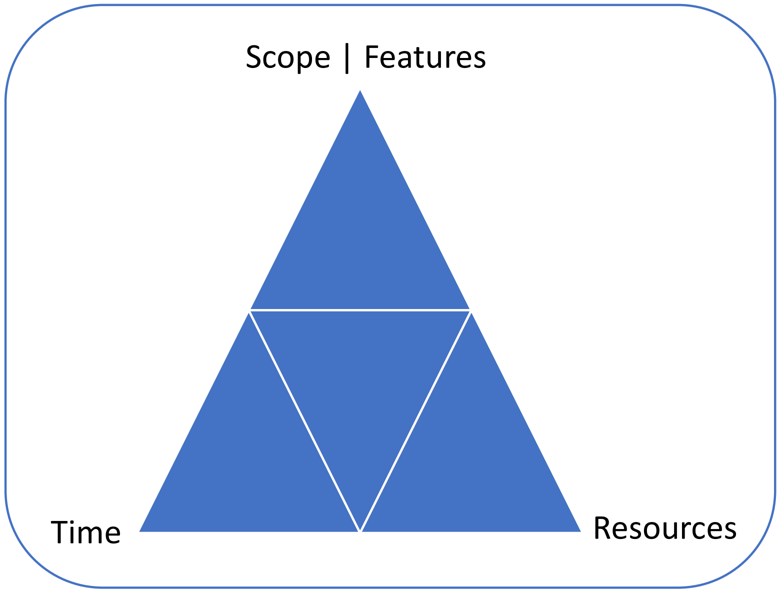
- Unclear scope leading to unclear requirements
- Finding the right balance (depends on customer, and the team):
  - Comprehensible vs. detailed
  - Graphics vs. tables and explicit and precise wording
  - Short and timely vs. complete and late
- Capturing implementation details instead of requirements
- Projecting your own models/ideas
- Feature creep



Feature creep?

For your project, consider **major** features and **stretch** features

Feature creep is the gradual accumulation of features over time, beyond what was originally committed and/or actually needed



Why does it happen? Because features are fun!

- Developers like to code them
- Sales teams like to pitch them
- Users (think they) want them

Why can it be bad?

- Can put your project delivery at risk
- Too many options, more bugs, more delays, less testing, …

Coming up on Friday

1. What are requirements and what is their value?
2. How can we gather requirements?
3. **What are techniques used to specify them?**
    - Use cases
    - Personas, user scenarios
    - Storyboarding
    - Paper prototyping
    - Prototyping
    - UML
    - …

Questions?